Building Multi-Agent Systems with Object-Oriented Language and Expert System Shell: A Process Control Application[1]

Li Wei
Software Engineer
Unit. 607 Bldg. 310 Zhong Guan Cun Nan 2 Tiao 1 Hao
Beijing 100080 P.R.China
8610-62986638-3767
liweid@legend.com.cn

Hongan Wang
Associate Researcher
Institute of Software, Chinese Academy of Sciences
Beijing 100080, P.R.China
wha@imd.cims.edu.cn

ABSTRACT

Agent programming has been received much attention recently in industrial application. In this paper we described a method for combining rule-based language with object-oriented languages, and established a multi-agent system for real-world problems. Based on the proposed multi-agent model, we developed a prototype system for the real-time supervision of oil transporting and storing in a refinery.

Keywords

multi-agent, object-oriented, rule, expert system shell

1. INTRODUCTION

Agent technology has been considered as an important approach for developing distributed intelligent manufacturing systems. Many researchers have attempted to apply agent technology to industrial application, such as manufacturing enterprise integration, supply chain management, manufacturing planning, scheduling and control, materials handling, and holonic manufacturing systems.

Among these published works, methods of building MAS systems are discussed. However, in this paper we focus on some practical issues of the MAS, and try to build our MAS based on the object-oriented system, which already exists.

Combining expert system shell (such as ILOG Rules, CLIPS, and Jess) with an object-oriented language, we give an example of multi-agent systems: Oil Storage Supervision and Decision Support System (OSS & DSS). The system has three fundamental features:

Firstly, OSS & DSS are assisted by a rules-base, which support agents to make quick and accurate decisions.

Secondly, OSS & DSS depend on acting in response to a dynamically changing environment.

Finally, the whole process was displayed by a graphical interface to demonstrate the changing status of oil storage and transport.

Section 2 describes a method to build practical multi-agent systems. Section 3 presents the architecture of a real-time multi-agent process system. Section 4 presented an example of real-time supervision of oil transporting and storing. Conclusions are given in Section 5.

2. CREATING INTELLIGENT OBJECT

A number of research projects and commercial offerings have addressed the Object System Embeddable Rule System [1, 2]. Rule groups can be defined to be objects and/or rules can be used to define individual members of an object. Embedding in an object system introduces many of the advantages of object-oriented programming to rule based programming. For example, embedding rules in objects introduces the control paradigm of message passing to rule-based programming.

But the following issues also arise when we apply rule-based expert system shell to object-oriented applications:

- How to add intelligent features to the old systems based on the object-oriented architecture and modules?
- How to keep the old applications working properly when they are switched to multi-agent systems from the object-oriented systems?
- How to keep high performance of the whole system to meet the real-time requirement, such as process control, network supervision (fraud detection), telecommunications and other areas that need real-time decision making?
- How to create a multi-agent system using rule-based expert system shell?

The major benefit of using rule-based expert system shell in object systems is clearly faster and easier code maintenance. This is due to:

1.Reasoning about objects with rules is easy and more natural. Rules are clearer and more maintainable than pure C++ or Java code.

2.The rules are separated from the core application source, which increases the system's modularity. Communication between the application and rules occurs through a simple API. This guarantees minimum impact on existing code when integrating

[1] This work is partly supported by National 95 Key Project of China, and by National Natural Science Foundation of China.

rules, while providing rules with maximum inter-operability with objects in application.

In our application, the communication bridge between the original object application and its rules is defined as some simple declaration.

Suppose we already have a C++ class, alarm, and we use it with rules:

```
enum PriorityType {low, high }
enum Color Type { white, red, green}
Class Alarm
{
private:
        ColorType          _color;
        PriorityType _priority;
public:
        Alarm() {}
        ~Alarm() {}
        const ColorType getColor();
        void setColor(ColorType color);
        const PriorityType getPriority() {};
        void setPriority (PrioriType priority)
}
```

Declare this class to rules-based expert system shell with the following implementation declaration:

```
(1)     (defimplementation Alarm()
(2)      (        (color type {ColorType}
(3)               reader {?object->getColor()}
(4)               writer {?object->setColor(?value)}
(5)               )
(6)               (priority type {PriorityType}
(7)               reader {?object->getPriority()}
(8)               writer{?object->setPriority(?value)}
(9)               )
(10)     )
(11)     constructor {new Alarm()}
(12)     destructor {delete ?object}
(13)     status external
(14)    )
```

Line1declares the Alarm class. The empty brackets indicate that the Alarm class is not inherited from another class. Line2 to 10 define the class attributes. For each one, give its C++ type and declare how to read and write their value. Lines 11 and 12 indicate how to create and destroy an object of this class. Line13 indicates that alarm objects are external.

The declaration above is the interface between iLog Rules and C++. Other rule-based expert system shells (Jess, JRules) have similar interface between themselves and embedded system. In iLog Rules every rule set leads to the generation of a C++ class, and several classes can therefore cooperate with an application, each class having its point of view of the application objects. In this way intelligent classes integrate naturally into object-oriented applications.

Figure 1 demonstrates the development cycle of an application. The interpretative capabilities of rule-based class enable developers to enter a faster development cycle.

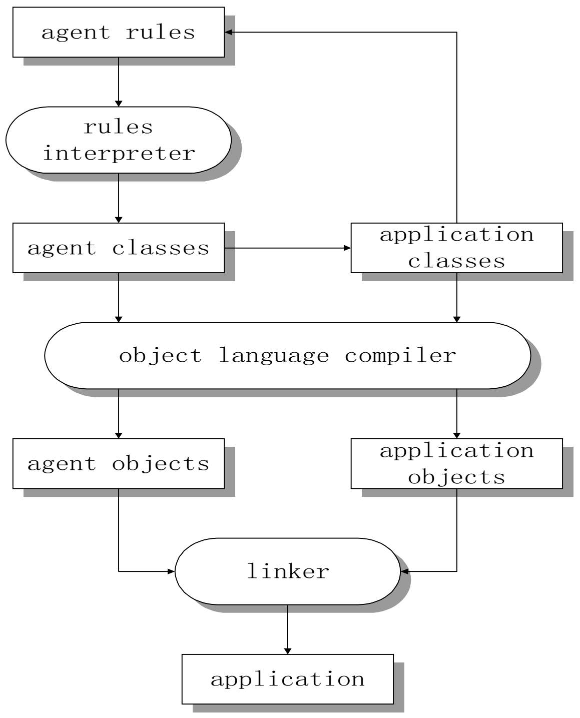

Figure 1

3. THE MULTI-AGENT MODEL

Multi-Agent Systems (MAS) is the subfield of Artificial Intelligence (AI) that aims to provide both principles for construction of complex systems involving multiple agents and mechanisms for coordination of independent agents behaviors. While there is no generally accepted definition of ``agent" in AI, for the purposes of this paper, an agent is object that can survey other application objects and behave intelligently according to them. An agent enables us to:

- To carry out fast multiple criteria search and join over the watched objects.
- Detect specific situations
- Prioritize actions

As we described above, the applications may be implemented with rule sets that detects the complex correlation of events and distills the relevant meaning from a large flow of data. So the rule set class play a role of agent. In other word, the agent is an instance of this class, some times the agent is a collection of several classes (include rule set object and other objects in application). The agent can then apply its rules to these objects by deducing and prioritizing actions according their status.

Figure2 demonstrates that at least one rule set object and some application objects (one or several) composed an agent. The agent includes several essential features:

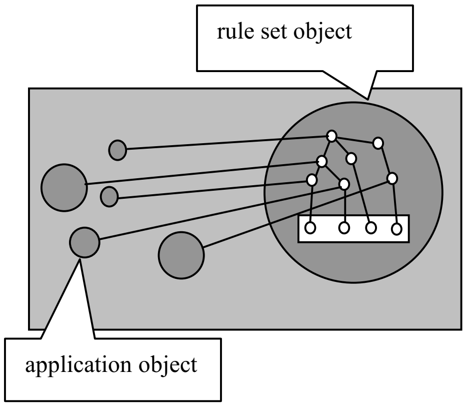

Figure 2

- The application objects through which the rule set object communicates with the application and other agent;
- The rule set object containing all the rules whose conditions are satisfied at a given moment (in the agenda, the rules are sorted according to their priority; for dynamic priority, a computation is performed when the rule is fired);
- Whether an application object is included in an agent is decided by the dependence of the object. If an application object can not be accessed by any rule set object, it will not be included in any agent. But in different situation, the result may be different.

The rule in fact encapsulated as a class and an agent is composed of an instance of that class and other objects. This means that we can handle agents as easily as we manipulate any object of C++ or Java.

Each compiled rule set is a class and is distinct from other rule sets. This independence enables us to use many different rule sets simultaneously, and therefore easily build multi-agent systems.

In our developed multi-agent systems, agents are divided by the different rule sets. Each agent has its own vision of the objects representing the function and knows different objects. For example, in a network supervision system, a network is made up a set of sites, each site itself being made up of a set of network elements. Each of these elements contains various internal cards. Each network element has its own rule agent to filter its internal alarm, and each site has its own rule agent to compute its status according to its element alarms. The simulator agent manages all these alarms. So a network element agent only knows the internal card of its element and the associated alarms, while the simulator agent knows all the internal cards and their alarms.

In the next section, we will build an actual multi-agent system example.

4. AN EXAMPLE OF PROCESS CONTROL

The example this section is an Oil Storage Supervision and Decision Support System (OSS&DSS).

The domain is composed of tanks, pumps, valves, and pipes. (Figure 3)The oil is divided into two types (Light Oil and Heavy Oil).

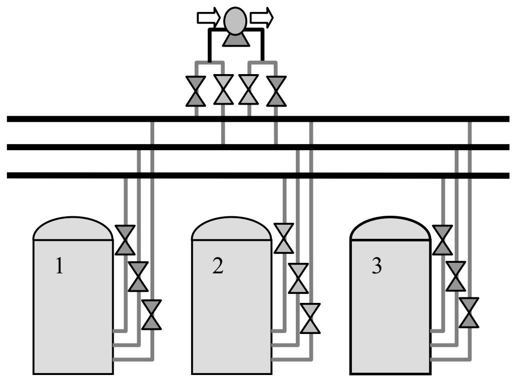

Figure 3

There are three types of agent in OSS&DSS: alarm agent, control agent and schedule agent.

The alarm agent performs the task of supervision. The rule set is below:

```
//The alarm detection of Liquidometer
(defrule TankFullAlarm high
         (CTank Liquidometer=?l1 Max=?l1 Flow>0 ID=?i)
         ?lp:(CLogicPath ToID=?i Status=ON)
        ->
(modify ?lp Status=OFF)
         (assert(CTankAlarm ID=?i Why=FULL))
         (activate Control)
         (activate Search)
)
        ...
```

The alarm agent collects the information of valve, pump, tank, pipe, and other application objects. If there are alarms on tanks, pumps and pipes, and only the tank alarms are significant. To avoid the manager being submerged by too many non-significant alarms, each element has its own rule set object to filter its own alarms determining which alarm is significant. Each element agent also computes its status according to its alarm. Then the status was provided to schedule agent to create suitable plan and avoid further alarm.

The schedule agents make plans according the follow rules set.

```
//find the Heavy Oil Tank to fill in
(defrule OutputHeavyOil default
(CTank ID=0 Content=HEAVY_OIL Flow=0)
    (CTank ID=?i1 & >0 & <4 Liquidometer>0
Content=HEAVY_OIL)
        ?lp:(CLogicPath FromID=?i1 ToID=0 LogicValveID=?i
        Weight=?lowest)
(not (CLogicPath FromID=?i1 ToID=0 Weight<?lowest))
            ->
    (modify ?lp Status=ON)
            (activate Control)
)
…
```

When schedule agent made a plan, the control agent must finish the tasks according to the plan. In the real world, the problem of producing timely responses when it faced with deadlines is an important issue for real-time control system. To deal with this problem different priority was assigned to each task produced by schedule agent. And control agent must build an acceptable way to solve the task considering the time available. The design-to-time scheduling will be discussed in another article. We just consider the common rules set of control agents.

```
//
(defrule TurnOffValve high
            (CLogicValve Status=OFF ID=?i)
(CPath LogicValveID=?i partName=VALVE partID=?pi
OffStatus=?s)
            ?v:(CValve ID=?pi Status<>?s)
->
            (modify ?v Status=?s)
            {?v->drawValve();}
)
```

The rules set above makes the control agent to turn off the valves according to the logic path, which is made by schedule agents.

The three kinds of agents communicate with each other and cooperate their behaviors via common objects. Figure 4 demonstrate the outline of multi-agent system of OSS&DSS.

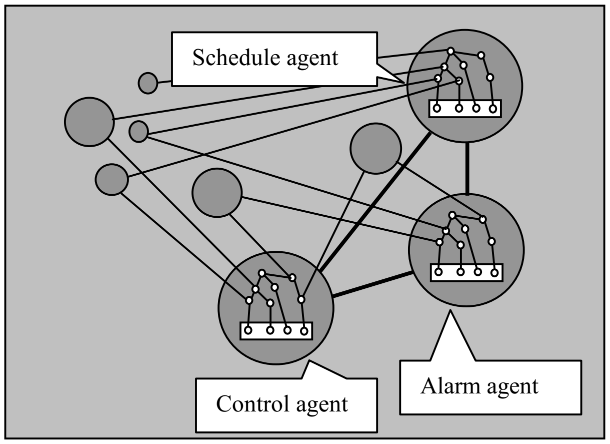

Figure 4

5. CONCLUSION

The paper shows how easily rule-based expert system allows us to design and develop multi-agent system that share the same implementation and communicate together in order to carry out a complex task. The synthesis between object-oriented programming and rule-based programming language paradigms makes the multi-agent system easy-to-develop.

6. REFERENCES

[1] Shlomo Zilberstein, Struart Russell, Optimal composition of real-time systems, in: Artificial Intelligence:, 82 (1996), 181-213.

[2] ILOG Rules White Paper http://www.ilog.com/

[3] Yoav Shoham. Agent-oriented programming, in: Artificial Intelligence:, 60 (1993), 51-92.